УДК 004.7:681.3

ОБРАБОТКА ПОТОКОВ ДАННЫХ В РЕЖИМЕ РЕАЛЬНОГО ВРЕМЕНИ: АРХИТЕКТУРА И ПРИМЕНЕНИЕ

Карпов Р.А.[1], студент гр. 6310, III курс
Научный руководитель: Палёнов Е.В.[2], ст. преподаватель
[1,2]КНИТУ-КАИ им. А.Н. Туполева
г. Казань

Представьте типичный пятничный вечер: миллионы людей расплачиваются картами, заказывают еду, переводят деньги. За каждым нажатием кнопки «оплатить» стоит цепочка событий, которую никто не видит, — и которая должна завершиться менее чем за 100 миллисекунд. Иначе пользователь почувствует паузу, а пауза — это уже потери. За эти 100 мс успевает произойти немало: запрос попадает в систему мониторинга мошенничества, строится граф связей между счётом, устройством и десятками параметров, ML-модель выдаёт оценку риска — и всё это на фоне тысяч параллельных транзакций. Вот что такое потоковая обработка данных на практике [1].

Данных становится больше — это банальность, но масштаб удивляет даже специалистов. По данным IDC, в 2023 году человечество сгенерировало около 120 зеттабайт информации. Значительная её часть рождается непрерывно: показания промышленных датчиков, клики пользователей, биржевые котировки, GPS-треки курьеров. Ценность многих из этих данных убывает почти мгновенно — температура двигателя, поднявшаяся до критической отметки три минуты назад, уже ничем не поможет предотвратить поломку. Именно поэтому потоковая обработка перестала быть экзотикой: она стала инженерной необходимостью [2].

От пакетной к потоковой обработке

Пакетная обработка — это логика фотоальбома: снимаешь целый день, вечером смотришь результат. Годами это было нормой: данные копились в хранилище, ночной скрипт запускал расчёты, утром аналитики получали отчёт. Никаких принципиальных проблем — пока задержка в несколько часов допустима. Но попробуйте построить антифрод-систему по такой схеме. К утренним отчётам мошенник давно обналичил деньги и успел покинуть страну [1].

Потоковый подход работает принципиально иначе: каждое событие обрабатывается в момент возникновения. Не «обработаем, когда накопится», а «обработаем сейчас». Это меняет не только технологию, но и способ мышления о данных: ответ на вопрос «что происходит?» становится важнее ответа на вопрос «что произошло?». Вместе с тем стоит оговориться: переход к потоковой архитектуре — решение не для всех. Если бизнес вполне

устраивает ежедневный отчёт, тратить ресурсы на потоковую систему попросту незачем [3].

Архитектура потоковых систем

Любая потоковая система, как бы она ни называлась, строится из одних и тех же компонентов. Источники генерируют события: датчики, мобильные приложения, базы данных через механизм CDC (Change Data Capture). Брокер — надёжный посредник, принимающий поток событий, буферизующий его и раздающий потребителям с нужными гарантиями доставки. Процессоры — слой, где данные обретают смысл: фильтрация, агрегация, обогащение из справочников, запуск ML-моделей. На выходе — хранилища, дашборды, уведомления или управляющие команды для других систем [2].

Как совместить потоковую скорость с точностью исторических расчётов? Этот вопрос породил Lambda-архитектуру, которая долго была де-факто стандартом. Идея: держать параллельно два слоя — потоковый для быстрых приближённых результатов и пакетный для точных исторических данных. Serving layer объединяет оба. Элегантно на бумаге, дорого на практике — по сути, нужно поддерживать две разные кодовые базы. Kappa-архитектура, предложенная Джейем Крепсом (одним из создателей Kafka), убирает пакетный слой вообще: историю переигрывают через тот же потоковый движок. Проще — но вычислительно дороже [3].

Apache Kafka: от утилиты LinkedIn до глобального стандарта

История Kafka начинается не в академической лаборатории, а в недрах LinkedIn примерно в 2010 году. Инженеры столкнулись с простой и неприятной задачей: десятки микросервисов генерировали события активности пользователей, и всё это нужно было надёжно собирать и передавать в реальном времени. Традиционные очереди сообщений — RabbitMQ, ActiveMQ — давали сбои при таких объёмах. Kafka создавалась как внутренний инструмент. Никто не рассчитывал, что через полтора десятилетия её будут использовать Netflix, Uber, Airbnb, половина крупных банков мира и тысячи других организаций [1].

Почему Kafka так хорошо масштабируется? Во многом из-за нескольких нетривиальных решений. Первое: данные хранятся на диске, а не в памяти — и это не медленно, потому что последовательная запись на диск быстрее случайного чтения из оперативной памяти. Второе: Kafka не удаляет сообщения после прочтения — она хранит их заданное время, и любой потребитель может «перемотать назад» и перечитать историю. Это меняет архитектурную модель: брокер становится не транзитным пунктом, а надёжным слоем хранения. Именно поэтому через Kafka в одном лишь LinkedIn сегодня проходит более 7 триллионов событий в сутки [1].

Высокую пропускную способность обеспечивает ещё один нетривиальный приём — zero-copy. При обычном чтении файла данные копируются из буфера диска в память ядра, затем в пространство пользователя, потом обратно в сетевой буфер. Kafka обходит эти промежуточные шаги, передавая данные напрямую из файловой системы в

сеть. На современном оборудовании это даёт сотни мегабайт в секунду на одном брокере, а горизонтальное масштабирование путём добавления новых партиций и брокеров не требует остановки кластера [1].

Фреймворки обработки: Flink, Spark и не только

Kafka решает транспортную задачу. Бизнес-логику нужно где-то выполнять — и здесь начинается мир фреймворков обработки. Apache Flink — пожалуй, самое мощное из доступных решений: он работает в по-настоящему потоковом режиме, обрабатывая каждое событие сразу, а не накапливая в мини-пакеты. Поддержка временны́х окон трёх типов — фиксированных, скользящих и сессионных, — stateful-обработка с гарантиями exactly-once, встроенные механизмы восстановления после сбоев — технически Flink закрывает большинство реальных сценариев. Alibaba использует его для обработки событий крупнейшего в мире торгового дня — китайского Дня холостяка [4].

Spark Structured Streaming идёт другим путём: технически это не «чистый» поток, Spark накапливает небольшие порции данных и обрабатывает их как мини-пакеты. Задержка — от нескольких сотен миллисекунд до нескольких секунд. Для задач, требующих реакции за десятки миллисекунд, этого может быть недостаточно. Но для команд, уже работающих со Spark, здесь нет ни нового API, ни новых инструментов отладки — а это весомый практический аргумент. Kafka Streams занимает ещё одну нишу: это библиотека без отдельного кластера, идеальная для сервисов, живущих целиком внутри экосистемы Kafka [4].

Где потоковая обработка действительно нужна

Антифрод — самый известный и убедительный сценарий. Visa и Mastercard обрабатывают тысячи транзакций в секунду через глобальные потоковые системы, и модель оценки риска видит не просто цифры транзакции — она видит контекст: время суток, страну, историю операций с конкретного устройства, сеть связанных счетов. Задержка принятия решения — десятки миллисекунд. Что важно: переход к потоковой архитектуре здесь дал не только прирост скорости, но и качественное улучшение — живой контекст вместо вчерашних данных снизил долю ложных срабатываний заметно [5].

Промышленный IoT — менее медийный, но экономически огромный сегмент. Производственные линии генерируют непрерывный поток данных с тысяч датчиков: температура подшипников, вибрация, ток двигателей, давление в трубопроводах. Пайплайн Kafka и Flink способен выявить аномальный паттерн с задержкой менее секунды — задолго до того, как сработают механические предохранители. По оценкам McKinsey, предиктивное обслуживание на основе IoT-аналитики сокращает незапланированные простои на 30–50%. Для крупного металлургического или нефтехимического предприятия это реальная экономия в сотни миллионов рублей в год [5].

В e-commerce потоковая обработка открывает персонализацию в реальном времени. Раньше рекомендательный движок строился на исторических данных: что пользователь покупал за последние месяцы. Это полезная модель, но она не видит то, что происходит прямо сейчас. Потоковая архитектура позволяет учесть текущую сессию целиком — какие страницы просматривались, в каком порядке, на чём задерживалось внимание, — и адаптировать выдачу под «сегодняшнего» пользователя, а не «исторического». В подобных экспериментах прирост кликабельности рекомендаций составляет, по данным разных компаний, 15–25% [5].

Что остаётся за кадром: сложность и цена

Потоковые системы не просто сложны — они очень сложны, и это стоит понимать до начала проектирования, а не в процессе. Тестирование временно́й логики требует специальных подходов: как проверить поведение системы при опоздавших событиях, если в тестовой среде время движется синтетически? Обработка late arrivals — событий, пришедших позже ожидаемого из-за сетевых задержек или перезапусков устройств, — это целый класс инженерных решений. Watermark'и, задающие момент «закрытия» временно́го окна, работают как компромисс между точностью и задержкой, и правильно настроить их для конкретной задачи нетривиально [4].

Гарантии доставки — ещё одна болезненная тема. At-least-once: данные не потеряются, но могут дублироваться. At-most-once: дублей нет, но события могут быть потеряны. Exactly-once: каждое событие обработано ровно один раз — звучит как то, что нужно всегда. Но на практике exactly-once требует транзакционной координации между брокером и обработчиком, что снижает пропускную способность на 10–30%. Платить такую цену везде нецелесообразно: там, где допустима at-least-once обработка с последующей дедупликацией, это будет заметная потеря производительности ради формальной гарантии [4].

Управление состоянием при масштабировании — третья по счёту, но не менее серьёзная трудность. Stateful-обработчики хранят рабочее состояние — например, суммы по скользящему окну — в локальных state store. При перераспределении нагрузки между узлами это состояние нужно мигрировать, что создаёт паузы и требует тщательного планирования топологии. Инструменты мониторинга и отладки потоковых пайплайнов существенно менее зрелы, чем аналоги для пакетной обработки, — это отдельный операционный риск [4].

Потоковая обработка данных прошла путь от нишевой технологии до базового инструмента инженерии данных. Kafka стала стандартом транспортного слоя, Flink и Spark Structured Streaming — основными инструментами обработки событий. Практическое применение охватывает финансовый сектор, промышленность, медиа и торговлю. Но главный вывод такой: потоковая архитектура — не «лучшая практика», которую следует применять по умолчанию. Это мощный инструмент с серьёзной ценой

владения, и его применение оправдано ровно там, где задержка действительно критична для конечного результата. Понять эту границу — значит сделать половину архитектурного решения правильно.

Список литературы:

1. Kreps J., Narkhede N., Rao J. Kafka: A Distributed Messaging System for Log Processing // Proceedings of NetDB Workshop. – Athens, 2011. – P. 1–7.
2. Самарев Р.С. Обзор состояния области потоковой обработки данных // Труды Института системного программирования РАН. – 2017. – Т. 29. – № 1. – С. 231–260.
3. Kleppmann M. Designing Data-Intensive Applications. – Sebastopol: O'Reilly Media, 2017. – 614 p.
4. Carbone P. et al. Apache Flink: Stream and Batch Processing in a Single Engine // IEEE Bulletin of the Technical Committee on Data Engineering. – 2015. – Vol. 38. – № 4. – P. 28–38.
5. Акимов А.В. Потоковые технологии обработки данных в задачах реального времени: обзор архитектурных решений // Программная инженерия. – 2022. – Т. 13. – № 4. – С. 178–186.